VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM KLUBU TANEČNÍHO SPORTU

INFORMATION SYSTEM OF THE DANCE CLUB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ OPICHAL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21180/2017/xopich00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Opichal Tomáš**
Obor: Informační technologie
Téma: **Informační systém klubu tanečního sportu**
Information System of the Dance Club
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s principy tvorby webových aplikací a potřebnými technologiemi.
2. Analyzujte požadavky na informační systém klubu tanečního sportu, který umožní plánování tréninků a individuálních lekcí včetně generování rozvrhu jednotlivých sálů, evidenci tanečních párů včetně informace o jejich umístění v žebříčku (bude se stahovat z webu CSTS.cz), hodnocení žáků i trenérů a možnost generování fuktur za indivuduální lekce.
3. Navrhněte informační systém dle výše uvedených požadavků, využijte k tomu vhodné modelovací techniky.
4. Navržený informační systém implementujte a ověřte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a navrhněte další možné pokračování tohoto projektu.

Literatura:

- Welling, L., Thomsonová, L.: PHP a MySQL: rozvoj webových aplikací. Vyd. 1. Praha: SoftPress, 2003, 910 s. ISBN 80-86497-60-7.
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015. ISBN: 978-80-251-4573-9

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce řeší problém implementace informačního systému, který by sloužil jako interní systém klubu tanečního sportu. Na základě požadavků na systém byl systém implementován pomocí PHP frameworku CodeIgniter. Mezi funkce systému patří zejména evidence událostí, generování dokladů za lekce nebo stahování potřebných dat ze stránek tanečního svazu. Systém úspěšně prošel základním testováním na skupině potenciálních uživatelů.

Abstract

This thesis solves the problem of the implementation of an information system, which could serve as the internal system for a dance club. Based on the system requirements, the system was implemented using the CodeIgniter PHP framework. In particular, system features include event logging, the generation of documents for the lessons and also the downloading of the necessary data from the dance union's website. The system successfully passed basic testing on a group of potential users.

Klíčová slova

Informační systém, taneční klub, PHP, CodeIgniter, MySQL, WAMP, MVC, rozvrh, události, API, PDF.

Keywords

Information System, dance club, PHP, CodeIgniter, MySQL, WAMP, MVC, schedule, events, API, PDF.

Citace

OPICHAL, Tomáš. *Informační systém klubu tanečního sportu.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Informační systém klubu tanečního sportu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Opichal
14. května 2018

Poděkování

Děkuji panu Ing. Vladimíru Bartíkovi, Ph.D. za poskytnutí odborné pomoci formou průběžných konzultací.

Obsah

Kapitola 1

Úvod

Tento dokument vznikl jako bakalářská práce na Fakultě informačních technologií Vysokého učení technického v Brně. Cílem této práce bylo vytvořit informační systém, který usnadní evidenci tanečních tréninků, účtování tanečních lekcí, zlepší informovanost členů tanečního klubu a celkově vnese do fungování tanečního klubu větší řád.

Již pátým rokem se věnuji společenskému tanci na soutěžní úrovni, jsem členem Klubu sportovního tance QUICK Olomouc. Během své taneční kariéry jsem měl možnost nahlédnout do zázemí klubů většího formátu, jako jsou taneční kluby v Ostravě nebo v Brně.

Taneční sport se postupně začíná dostávat do povědomí veřejnosti, kluby nabírají stále nové a nové členy, taneční tréninky probíhají na nejrůznějších prostorech, celkově vzato, přišel čas, kdy je třeba začít tyto informace *evidovat*. Vzhledem k tomu, že jsem přímo součástí tanečního klubu a dostávám se často do kontaktu s vedením, jsem schopen velmi efektivně určit požadavky na výsledný systém.

Kapitola *Použité technologie* představuje teoretickou část tohoto dokumentu, čtenář se zde seznámí se všemi důležitými technologiemi, které byly v rámci mé práce použity. Další kapitola nese název *Úvod do problematiky*. Tato kapitola obsahuje seznámení s pojmy z tanečního sportu a vysvětluje, proč je nutné zabývat se vývojem systému pro taneční klub. V kapitole *Návrh* lze najít výsledky fáze návrhu informačního systému včetně návrhu uživatelského rozhraní. Kapitola *Implementace* obsahuje vysvětlení postupů, které byly použity při samotné implementaci systému se zaměřením na problémy, které jsou specifické pro daný informační systém. V kapitole *Testování* je vysvětleno, jakým způsobem byl systém prověřen kvůli odhalení případných nedostatků.

Kapitola 2

Použité technologie

Tato kapitola je věnovaná jednotlivým technologiím, se kterými jsem se musel seznámit za účelem vytvoření informačního systému.

2.1 HTML

HTML, neboli *HyperText Markup Language*, je název značkovacího jazyka, který je zpracováván na straně uživatele systému, nejčastěji ve webovém prohlížeči. Jazyk HTML je založen na univerzálním značkovacím jazyku SGML, velice komplexním jazyku, jehož složitost mu zabránila ve větším rozšíření.

První specifikace HTML 1.0 se objevila již v roce 1991. Postupně se vývojem HTML dostalo v roce 1999 na verzi 4.01, což měla být podle původních předpokladů poslední verze HTML, které mělo být kompletně nahrazeno XHTML. To se ale nestalo, důvodem byla především iniciativa pracovní skupiny **WHATWG**, jejíž vznik byl odpovědí na pomalý vývoj webových technologií konsorcia **W3C**.

Základními prvky jazyka HTML jsou *tagy*, jejichž prostřednictvím lze definovat vlastnosti výsledného dokumentu. Jednotlivé tagy mají speciální sémantický význam. To je znázorněno ve fragmentu 2.1.

```
<b>Bold text.</b>
<b><i>Bold and italic text.</i></b>
```

Fragment kódu 2.1: Ukázka HTML

První řádek fragmentu zvýrazní tučně text, který se nachází mezi počátečním a koncovým tagem. Text na druhém řádku bude interpretován jako tučný a kurzívou. [11]

Hlavní úlohou jazyka HTML (v kontextu informačních systémů) je poskytnout uživateli rozhraní pro práci nad daty systému. HTML podporuje různé druhy zobrazení, například pomocí *tabulek* nebo *seznamů*. Pomocí HTML je také možné definovat formuláře, které umožňují uživateli předat systému nová data.

2.2 CSS

CSS je zkratkou pro *Cascading Style Sheets*. Jedná se o jazyk, který umožňuje popis vzhledu HTML dokumentů. Snahou vývojářů jazyka CSS bylo oddělit vzhled dokumentů od jejich

struktury a obsahu. Starší verze HTML umožňovaly měnit více způsob zobrazení obsahu dokumentu. Dnes se ale již dá tvrdit, že CSS tuto funkcionalitu HTML zcela nahradilo.

CSS se zpravidla definuje jako seznam *pravidel*. Každé pravidlo sestává ze *selektoru* a *deklaračního bloku*. Selektor slouží k určení, které prvky HTML dokumentu budou pravidlem modifikovány. V deklaračním bloku následuje seznam dvojic *vlastnost* - *hodnota*, které definují vzhled konkrétního prvku. [3] Zápis jednoho CSS pravidla můžete vidět na fragmentu 2.2.

```
div.black {
        color: white;
        background: black;
}

<div class="black">
        Black background, white text.
</div>
```

Fragment kódu 2.2: Ukázka CSS

2.3 JavaScript

Jedná se o multiplatformní, objektově orientovaný, skriptovací jazyk, jehož podpora je integrovaná přímo do prohlížeče na straně uživatele. Z tohoto hlediska plynou jistá bezpečnostní omezení. JavaScript na straně uživatele například nemůže pracovat se soubory.

Na druhou stranu však poskytuje řadu možností, jak přizpůsobit uživatelské prostředí. Jedním z nejčastějších použití je vykonávání funkcí na základě takzvaných *onclick událostí*. Ty nám umožňují například zobrazit určitý obsah stránky až ve chvíli, kdy si jej uživatel vyžádá. [9] **Nejedná** se ovšem o odesílání této žádosti pomocí HTTP dotazů - pro tyto účely slouží technologie *AJAX*, kterou ovšem v mé implementaci systému nevyužívám. Obsah je již tedy kompletně dostupný na straně uživatele a ten si pouze vybírá, který obsah má být v daný moment viditelný.

2.3.1 jQuery

jQuery je javascriptová knihovna, která cílí na zjednodušení psaní klientských skriptů nad HTML dokumentem. Ne náhodou je tato knihovna zdaleka nejpoužívanější javascriptovou knihovnou při vývoji webových aplikací.

Syntaxe jQuery je navržená tak, aby zjednodušovala především vybírání HTML elementů, vytváření animací a zpracování událostí. Výběr HTML elementů připomíná syntaxi jazyka CSS, proto je psaní v jQuery daleko intuitivnější než v čistém JavaScriptu. [7]

Mezi principy jQuery patří zejména

- oddělení JavaScriptu a HTML,
- stručnost a srozumitelnost,
- zvýšení kompatibility napříč prohlížeči,
- rozšiřitelnost.

2.4 PHP

Vzhledem k tomu, že tento jazyk v implementaci systému jednoznačně dominuje, dovolím si této sekci věnovat o něco větší pozornost než ostatním.

PHP se řadí mezi dynamicky typované jazyky, to znamená že datový typ proměnné je vázán na hodnotu proměnné, nikoli na samotnou proměnnou. Tento přístup má samozřejmě své výhody, ale také svá úskalí. Obecně lze říci, že pokud člověk píše v PHP, musí být ostražitější, protože ze zřejmých důvodů není možná statická kontrola datových typů. Na druhou stranu, psaní kódu v PHP je díky dynamickému typování pružnější, což často vede k rychlému dosažení cíle.

2.4.1 Kousek historie

Zrod PHP se datuje do roku 1995. Tehdy jistý vývojář jménem *Rasmus Lerdorf* vytvořil na svou dobu průlomový skript (Perl/CGI), který umožňoval protokolovat informace o návštěvníkovi a zobrazoval počet návštěvníků na webu. Rázem Lerdorfovi začaly chodit e-maily s žádostmi o jeho skripty. Ten je začal dávat ostatním k dispozici pod hlavičkou *Personal Home Page*.

V roce 1997 spatřila světlo světa verze PHP 2.0. Tato už byla ovlivněna různými zdokonaleními od programátorů po celém světě. Ve stejném roce došlo také k přejmenování PHP na *Hypertext Preprocessor*. Verze 3.0 přišla hned o rok později, tou dobou již PHP aktivně využívalo přes 50 000 uživatelů. Roku 2002 byla vydaná verze 4.0, která je považovaná za oficiální debut PHP na vývojářské scéně korporací. Dva roky na to přinesla verze 5.0 velké zdokonalení v oblasti OOP. Verze 6.0 se pokusila přinést do PHP nativní podporu kódování Unicode, bohužel se však dostala do slepé uličky. A nakonec verze PHP 7.0 přinesla přepsání mechanismů pro zotavení z chyb na systém výjimek a také změnu syntaxe v oblasti dereferencování některých typů proměnných. [4]

2.4.2 Rysy jazyka

V této kapitole zmíním důvody, které dělají PHP tak populárním, přestože se najde celá řada vývojářů, kteří se na tento jazyk dívají skrz prsty. [4]

Upotřebitelnost

Záměrem pana Lerdorfa bylo především vytvořit nástroje, které dokážou řešit konkrétní úkoly, pro které ještě neexistuje řešení. Proto se celý vývoj jazyka PHP soustředil na zlepšení použitelnosti ze strany uživatele jazyka. Tato strategie je hlavním důvodem, proč se dá PHP nazvat *minimalistickým jazykem* z pohledu kombinace požadavků uživatele a syntaxe jazyka. Dynamická typová kontrola tuto vlastnost skvěle podporuje.

Vyspělost

Přestože PHP cílí především na vytváření dynamických webových stránek a nabízí tedy podporu práce s databází nebo zpracování formulářů, nezůstalo pozadu ani z pohledu podpory obecně používaných technologií v programátorském světě. Konkrétně se jedná o:

- práce se soubory obrázků a PDF,
- komunikace s LDAP,

- pokročilá podpora regulárních výrazů
- komunikace prostřednictvím nejrůznějších protokolů (IMAP, POP3, DNS, ...)
- komunikace s různými řešeními pro zpracování kreditních karet

Možnosti

Dalším důvodem popularity PHP je možnost volby. Už jen fakt, že PHP nabízí podporu minimálně pětadvaceti databázových produktů (samozřejmě včetně populárních *MySQL*, *Oracle* nebo *PostgreSQL*) stojí rozhodně za zmínku. Dále nesmíme zapomenout na podporu dvou různých formátů regulárních výrazů kompatibilních s POSIX, případně s Perlem a v neposlední řadě možnost volby mezi klasickým *procedurálním programováním* a *objektovým přístupem*.

2.5 Relační databáze

Databází rozumíme kombinaci **dat** a **softwarových prostředků**, které umožňují s daty manipulovat a přistupovat k nim. Takové softwarové prostředky často označujeme jako *systém řízení báze dat*.

Relační databází rozumíme databázi, která používá principy *relačního datového modelu*.

2.5.1 Relační datový model

Jedná se o způsob uložení dat v relačních databázích. Základními pojmy tohoto modelu jsou: [5]

- **Relace** - databázová tabulka, skládá se ze *schématu relace* a *těla relace*. Schéma je definováno jako $R = R(A_1 : D_1, A_2 : D_2, \ ... \ A_n : D_n)$, kde $A_i (A_i \neq A_j$ pro $i \neq j)$ označuje jméno atributu definovaného na doméně D_i. Tělo je definováno jako $R^* \subseteq D_1 \times D_2 \times ... \times D_n$. Počet atributů relace se označuje jako *stupeň (řád) relace*, kardinalita těla relace $m = |R^*|$ se označuje *kardinalita relace*. Matematický zápis byl převzat z [5].
- **Doména** - obor hodnot jednoho *atributu*, v praxi často definovaná *datovým typem*. Domény mohou obsahovat pouze *atomické* hodnoty, proto je nutné vhodně zvolit jednotlivé atributy. Někdy nelze obecně určit, co je atomickou hodnotou, např. adresa může být někdy atomickým atributem, jindy je vhodnější ji rozdělit na město, ulici atd.
- **Atribut** - jedna vlastnost relace, názvy atributů jsou často v záhlaví tabulky, které se souhrnně označuje jako *schéma relace*.

2.5.2 MySQL

MySQL je systém řízení báze dat, který uplatňuje relační datový model. Pro přístup k datům a jejich modifikaci používá dotazovací jazyk SQL.

MySQL server se těší své oblíbenosti především díky velmi dobré výkonnosti. Té dosahuje zejména díky kombinaci různých technologií, např. možnosti volby zpracovatele tabulek

(InnoDB, MyISAM, HEAP), *cachování dotazů* nebo *fulltextovému indexování a vyhledávání*.

InnoDB

V současné době je InnoDB výchozím formátem úložiště dat v MySQL databázích. Mezi jeho klíčové vlastnosti patří podpora *cizích klíčů*, která umožňuje kontrolu vztahů mezi tabulkami při modifikaci dat v databázi. Užitečná je také podpora *transakcí*, která zajišťuje, aby se SQL příkazy chovaly jako atomické operace. Tato vlastnost přispívá k zachování konzistence napříč databázovými tabulkami, jedná se o důležitou schopnost při správě citlivých dat jako jsou například finance.

Z hlediska výkonnosti je velice dobrou vlastností InnoDB také uzamykání na úrovni řádků tzv. *row-level locking*, tedy při modifikaci záznamů v databázi se zamyká ostatním uživatelům pouze konkrétní řádek, nikoli celá tabulka. V neposlední řadě je třeba zmínit, že tabulky InnoDB jsou schopny se obnovit, vznikne-li havárie. [4]

2.6 MVC

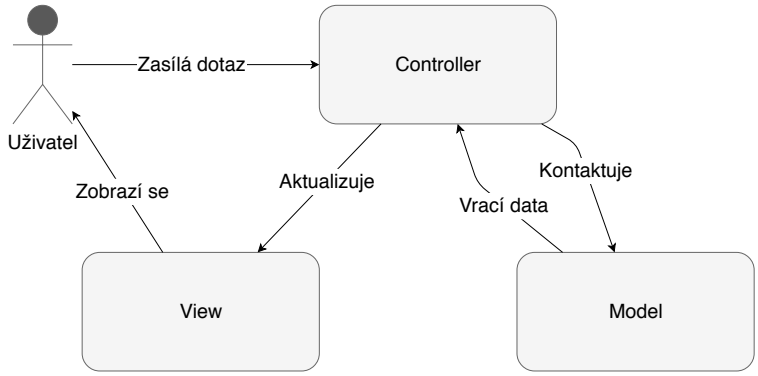
Model-view-controller je architektonický vzor, velice často používaný nejen při vývoji webových aplikací, ale také například v aplikacích pro operační systém **iOS**.

MVC pomáhá vývojáři především s oddělením jednotlivých částí softwaru, kdy každé části předává zodpovědnost za něco jiného.

Model je určen pro přístup k datům aplikace. Data jsou často uložená v databázi, se kterou model komunikuje a data z ní získává nebo je modifikuje. Navíc poskytuje rozhraní, pomocí kterého lze jednotlivé úkony nad daty provádět.

Controller je často považován za jádro celé aplikace. Zpracovává požadavky od uživatele a je-li to třeba, kontaktuje model. Další zodpovědností controlleru je, *co* se zobrazí uživateli (nikoli však jakým způsobem). V controlleru se často vyskytuje logika, která kontroluje oprávnění uživatele příslušné akce provádět. Na rozdíl od modelu si controller tedy vybírá, co uživateli zobrazí a přesně to si od modelu vyžádá.

View představuje vizuální složku celé aplikace. Obvykle se zde vyskytuje část kódu, která udává strukturu zobrazeného souboru, do které jsou zaneseny hodnoty proměnných, dodané z controlleru. View by měl komunikovat výhradně s *controllerem*, dle mého názoru se však občas vyskytnou situace, kdy je výhodné zavolat přímo model. Ve view je vhodné používat *šablonovací systém*, kdy je statický obsah stránky oddělen od dynamicky se měnícího obsahu. Takový přístup eliminuje vznik redundantního kódu a je nesrovnatelně jednodušší jej spravovat.



Obrázek 2.1: Tok událostí v MVC

2.7 CodeIgniter framework

Framework je softwarová struktura, která usnadňuje vývoj softwarového produktu tak, že nabízí nástroje, které řeší typické problémy dané oblasti. Tvůrce PHP Rasmus Lerdorf je však známým odpůrcem dnešních PHP frameworků, zmiňuje 2 důvody, proč se používání především tzv. *general purpose* frameworků (pro obecné účely) raději vyhnout:

1. Frameworky vykonávají stejný kód opakovaně, aniž by to bylo třeba.
   Pan Lerdorf toto uvedl na příkladu, kdy s každým HTTP dotazem framework opakovaně zjišťuje, jaký typ databáze aplikace používá.

2. Frameworky potřebují příliš mnoho na sobě závislých tříd.
   Tohle se ukazuje jako problém v případě, kdy chce programátor použít pouze část frameworku, kterou potřebuje. Často v takových případech narazí na problém závislostí napříč celým frameworkem.

Pan Lerdorf nehaní frameworky v obecné rovině. Zmiňuje, že pokud je framework zaměřený na konkrétní cíl a nemá spoustu zbytečné funkcionality navíc, nevidí důvod proč ho nevyužít. Jako příklady uvádí *Wordpress* pro uživatele, kteří zabývají tvorbou blogů, nebo *Drupal* pro ty, kteří chtějí mít větší kontrolu nad správou obsahu.

Abych uzavřel myšlenky pana Lerdorfa, dovolím si na závěr jednu citaci. Rasmus Lerdorf řekl, že má rád CodeIgniter, *„protože je rychlejší, méně komplikovaný a nejméně připomíná framework"*. [6]

2.7.1 Organizace souborů

CodeIgniter poskytuje vývojářům dobře připravenou adresářovou strukturu, která zároveň podporuje použití návrhového vzoru Model-view-controller. Pro každou komponentu z MVC architektury je zde samostatný adresář (s odpovídajícími názvy *models*, *views* a *controllers*). Tyto adresáře lze dále libovolně strukturovat dle vlastní potřeby.

2.7.2 Používání komponent

Načítání modelu

Jak lze v CodeIgniteru načíst a použít existující model je znázorněno na fragmentu 2.3.

```php
<?php
        $this->load->model('CarModel');
        $redCars = $this->CarModel->getCarsByColor('red');
```

Fragment kódu 2.3: Použití modelu

Uvedený kus kódu nám zpřístupní metody modelu *CarModel* a do proměnné nám uloží všechna červená auta. Obvykle se tato logika děje v controlleru.

Zobrazení view

CodeIgniter umožňuje dva způsoby, jak zobrazit soubory typu view.

- Prosté zobrazení, v podstatě ekvivalentní s PHP příkazem *include*.
- Zobrazení s předáním dat. Tohle je obvyklejší scénář.

Uvedu pouze příklad s předáním dat. Data lze předávat dvěma způsoby, a to prostřednictvím *asociativního pole* nebo prostřednictvím *objektu*. Ve fragmentu 2.4 můžete vidět metodu předávání pomocí asociativního pole.

```php
<?php
        $data['cars'] = $redCars;
        $this->load->view('carEngineView', $data);
```

Fragment kódu 2.4: Načtení view souboru

V souboru, *carEngineView.php* by byla data zobrazena například tak, jak ukazuje fragment 2.5.

```php
<?php
        foreach($cars as $car){
                echo '<h2>' . $car->name . '</h2>';
                echo '<ul>';
                echo '<li>' . $car->engineType . '</li>';
                echo '<li>' . $car->enginePower . '</li>';
                echo '</ul>';
        }
```

Fragment kódu 2.5: Zobrazení dat ve view souboru

Všimněte si tedy, že ač jsou data předávaná jako asociativní pole, lze k nim ze strany view přistupovat jako k obyčejným proměnným, jejichž názvy odpovídají názvům klíčů v původním asociativním poli, potažmo názvům vlastností původního objektu.

2.7.3 Licence MIT

CodeIgniter je od verze 3.0 uvolňován pod MIT licencí.

Tato svobodná licence vznikla v Massachusettském technologickém institutu (Massachusetts Institute of Technology, MIT). Software, který je distribuován s touto licencí je možné libovolně měnit a použít i ke komerčním účelům za podmínky, že v produktu bude uvedeno licenční upozornění s odpovídajícím copyrightem. Zároveň je třeba myslet na to, že je tento software distribuován bez jakékoli záruky. [10]

2.8 WampServer

Abych při programování nebyl závislý na hostingu a mohl systém vyvíjet lokálně, použil jsem technologii WampServer (http://www.wampserver.com/). Jedná se o softwarový balíček určený pro operační systémy Windows. Jeho hlavními komponentami jsou *Apache Web Server*, databázový systém MySQL a interpret jazyka PHP.

O MySQL a PHP již byla řeč. Úlohou web serveru je jednoduše řečeno vyřídit HTTP požadavky specifikované pomocí URL. Web server tedy funguje jako spojka mezi URL a controllerem (v architektuře MVC).

Kapitola 3

Úvod do problematiky

V následující kapitole bych rád nastínil důvody, které mě vedly k myšlence vytvořit informační systém tohoto typu a seznámil čtenáře se základními pojmy z tanečního světa.

3.1 Základní pojmy

Taneční klub - skupina lidí, kteří se věnují tanci na soutěžní úrovni, členy tanečního klubu lze zpravidla rozdělit na *tanečníky* a *trenéry*.

Taneční sál - prostor, na kterém probíhají taneční tréninky. Může se jednat o majetek tanečního klubu, nebo se prostor pronajímá. Prostorem může být například *tělocvična* nebo *sál kulturního domu*.

Skupinový trénink - jedná se o typ tréninku, kdy trenér vede skupinu lidí. Zpravidla se opakuje každý týden. Skupinový trénink lze dále rozdělit na *seminář* a *practise*.

Seminář - typ tréninku, kdy trenér učí skupinu lidí stejnou dovednost.

Practise - typ tréninku, který slouží jako simulace reálné taneční soutěže. Trenér je zde v roli pozorovatele a dává tanečníkům zpětnou vazbu.

Individuální lekce - typ tréninku, kdy trenér učí jednotlivce nebo taneční pár.

3.2 Informovanost členů

Za nejzávažnější problém fungování dnešních tanečních klubů považuji absenci jednotného zdroje informací. Nejeden taneční klub používá jako hlavní komunikační kanál skupinu na sociální síti Facebook, která ovšem pro potřeby tanečního klubu rozhodně nestačí. Informace v takové skupině jsou velmi obtížně dohledatelné, chybí strukturovanost a šance na rozšíření tohoto pseudosystému je nulová.

Členové tanečního klubu by měli mít vždy možnost dohledat, kdy je taneční sál dostupný k volnému tréninku, kdy je na sále vedený trénink nebo kdy bude na sále nejvíce místa. Kromě těchto informací je neméně důležité dát všem členům klubu vědět v případě, že se objeví výjimečná událost, která může mít za příčinu například zrušení nebo omezení

tréninku. Pokud členové tuto informaci nedostanou, dochází k nepříjemným situacím, kdy například tanečník přijde na trénink a zjistí, že se v tělocvičně koná florbalový turnaj.

3.3 Zjednodušení procesu účtování

Tréninky v tanečním klubu samozřejmě musí vést školený trenér s platnou licencí. Tuto činnost je možné provozovat na živnostenský list. V praxi to vypadá tak, že skupinové tréninky jsou trenérovi vypláceny z klubových peněz. Pokud si ale jednotlivec nebo taneční pár domluví s trenérem individuální lekci, je potřeba lekci zaplatit přímo trenérovi. Tanečníci mohou vyžadovat vydání dokladu o zaplacení, což samozřejmě pro trenéra znamená být vždy připraven doklad vydat. Cílem informačního systému by mělo být zjednodušení procesu vydání takového dokladu.

3.4 Motivační nástroj

Členové taneční komunity jsou obecně velice soutěživí, jsou to koneckonců sportovci, všichni chtějí vyhrávat, všichni chtějí být na vrcholu. Proto je vhodné do informačního systému zavést motivační nástroj, který bude tanečníkům dávat možnost postoupit v jakési klubové výkonnostní hierarchii. Další funkcí informačního systému tedy bude zobrazení výkonnostního žebříčku a také získávání anonymní zpětné vazby z uskutečněných individuálních lekcí.

Seznam faktorů, které mohou pozitivně i negativně ovlivnit zpětnou vazbu z individuální lekce:

- včasný příchod na lekci
- fyzická připravenost na lekci (únava, roztančení)
- *dress-code* účastníků lekce (tanec je estetický sport)
- vzájemný respekt mezi účastníky lekce
- *entuziasmus* (nadšení) účastníků lekce

3.5 Požadavky na systém

Na základě vlastního uvážení v kombinaci s rozhovory se členy našeho tanečního klubu jsem dal dohromady požadavky na výsledný systém. Systém není určen běžné veřejnosti, jakákoli data systému budou dostupná až na základě autentizace uživatele. Systém bude nabízet čtyři uživatelské role - *tanečník*, *trenér*, *hlavní trenér*, *administrátor*. Samovolná registrace bude taktéž zakázána, členové budou vytváření administrátorem. Systém musí obsahovat logiku pro spárování dvou tanečníků a evidenci jejich bodů v žebříčku, který bude možné *aktualizovat* pomocí datového přenosu z webu Českého svazu tanečního sportu (www.csts.cz).

Jádrem systému budou pak jednotlivé události, které se odehrávají v konkrétních klubových prostorech. Bude možné události vytvářet, upravovat a filtrovat. Systém musí umožňovat vytváření tzv. **periodických událostí**, tedy událostí, které se opakují každý týden. Takové události budou omezeny počátečním a koncovým datem. Musí být v případě potřeby umožněno modifikovat pouze konkrétní událost. Správu událostí a prostor bude mít na starosti hlavní trenér.

Součástí systému bude také evidence individuálních lekcí, které mohou být nabídnuty jak tanečníkem (trenérovi), tak i opačným směrem. Každou nabídnout lekci musí schválit taktéž druhá strana. V případě, že trenér nabídne lekci tanečnímu páru, stačí, aby lekci schválil jeden člen páru. Lekce bude možné zpětně hodnotit a poskytnout tak tanečníkům a trenérům *anonymní* zpětnou vazbu. Z důvodu zachování anonymity bude hodnocení agregováno a zobrazeno formou průměrného hodnocení za delší časový úsek.

Veškeré události, včetně individuálních lekcí budou základem pro *generování rozvrhu*. Rozvrh bude svázán s konkrétním prostorem. Systém bude poskytovat nástroje pro definování tréninkových časů na každém prostoru zvlášť. Jedná se o časový interval, kdy je prostor tanečníkům dostupný (v případě pronajímání prostor může být takový interval značně omezený). Prostor musí rovněž obsahovat informaci o tom, kolik zde lze uskutečnit individuálních lekcí najednou. Není přípustné, aby na prostoru o velikosti 5 x 5 metrů bylo vyučováno například 3 a více individuálních lekcí.

Tanečníci budou mít možnost nechat si vygenerovat příjmový doklad za každou individuální lekci, v případě, že trenér vyučuje na základě živnostenského oprávnění. Není třeba evidovat jednotlivé doklady, jejich evidenční číslo (a podpis) bude doplněno jako jediný z údajů až po vytištění. Trenér totiž může vydávat doklady i jinde, automatické číslování dokladů tedy není realizovatelné.

3.5.1 Bodování párů v rámci svazu

Pro lepší pochopení výsledné implementace žebříčku párů krátce vysvětlím bodovací systém v rámci Českého svazu tanečního sportu (dále jen *svaz*). Pro zachování jednoduchosti vynechám popis jednotlivých věkových kategorií.

Svaz uchovává o registrovaných tanečnících určitá data, která bude pro potřeby našeho systému nutno získávat. K *identifikaci* jednotlivých tanečníků nám poslouží číslo **IDT**, které je jedinečným identifikátorem tanečníka v rámci svazu.

Bodování tanečních párů lze obecně rozdělit do dvou rovin:

- bodování postupových soutěží
- bodování Ranklistu ČR

Bodování postupových soutěží

Pokud si nově složený pár opatří soutěžní licenci, automaticky je zařazen do systému *postupových soutěží*. V závislosti na předchozích výsledcích obou členů páru je pár zařazen do určité *výkonnostní kategorie*. Jednotlivé kategorie se označují písmeny E, D, C, B, A, M, kde kategorie E je výkonnostně nejnižší a kategorie M je maximální dosažitelná kategorie.

Až na výjimky se taneční pár účastní soutěží své kategorie a získává *body* a *finálová umístění*. Počet získaných bodů z jedné soutěže závisí na počtu poražených párů. Finálová umístění mají trochu zavádějící název, protože i když pár postoupí do finálového kola, nemusí mít ještě zisk finálového umístění jistý (např. finále se účastní šest párů, ale finálové umístění získají pouze první čtyři páry).

Jakmile pár získá potřebný počet bodů a finálových umístění, automaticky postupuje o jednu kategorii výše. V kategorii M se již postupové soutěže nepořádají.

Bodování Ranklistu ČR

Ranklist ČR je jakýmsi ukazatelem *aktuální* výkonnosti tanečního páru, protože se do něj počítají pouze soutěže za poslední rok.

Dobrým přirovnáním pro Ranklist ČR může být například žebříček ATP ve světě tenisu.

Do Ranklistu ČR lze získávat body třemi způsoby:

1. účastí na soutěžích *Taneční ligy*
2. účastí na soutěžích typu *Open*
3. účastí na Mistrovství České republiky

Hlavním významem Ranklistu ČR je zajištění nominace tanečního páru na aktuální ročník Mistrovství České republiky. Body do tohoto žebříčku se tedy snaží získávat zejména taneční páry, které již zaznamenaly určitý postup v hierarchii postupových soutěží.

Taneční ligy jsou typy soutěží, které jsou otevřené párům od určité výkonnostní kategorie (hraniční kategorie se liší napříč *věkovými kategoriemi*). Tyto soutěže se konají na území České republiky a jsou určeny výhradně českým tanečním párům.

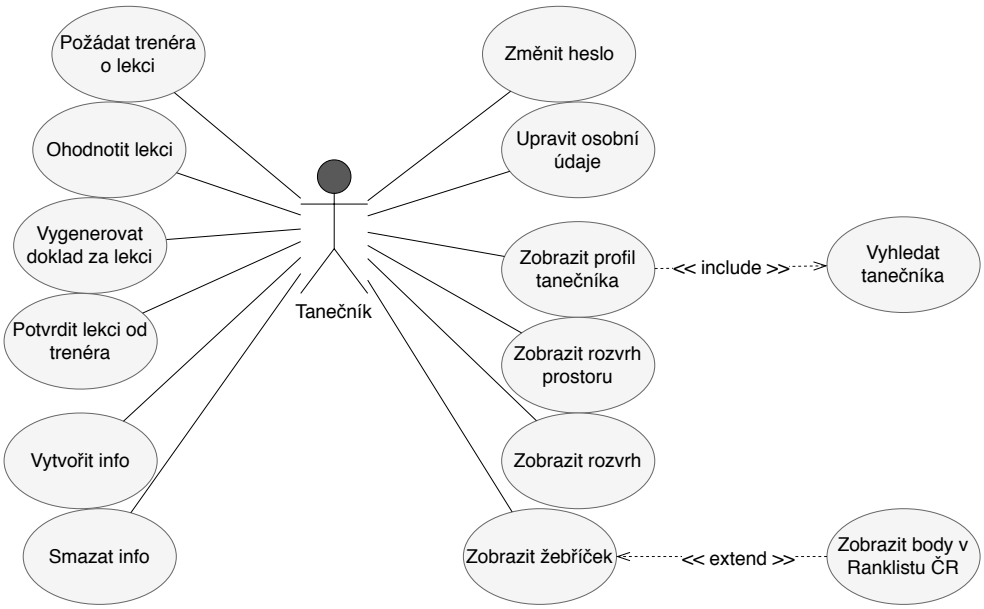
Soutěže typu Open se konají pod hlavičkou *World DanceSport Federation*. V rámci mezinárodní federace se lze účastnit také zahraničních soutěží. Tyto soutěže jsou určeny především párům, pro které domácí Taneční liga již nepředstavuje tak velkou výzvu.

Skupiny tanců

Společenský tanec na soutěžní úrovni aktuálně nabízí *dvě* skupiny tanců, ve kterých lze soutěžit, a těmi jsou tance **latinsko-americké** a **standardní**. Každá z těchto skupin zahrnuje pět konkrétních tanců. Výjimku v tomto systému představuje Mistrovství České republiky, které se navíc pořádá také v kategorii **deseti tanců**. Toto mistrovství je určeno tanečním párům, které mají odpovídající výsledky v obou jednotlivých skupinách tanců.

3.6 Role tanečníka

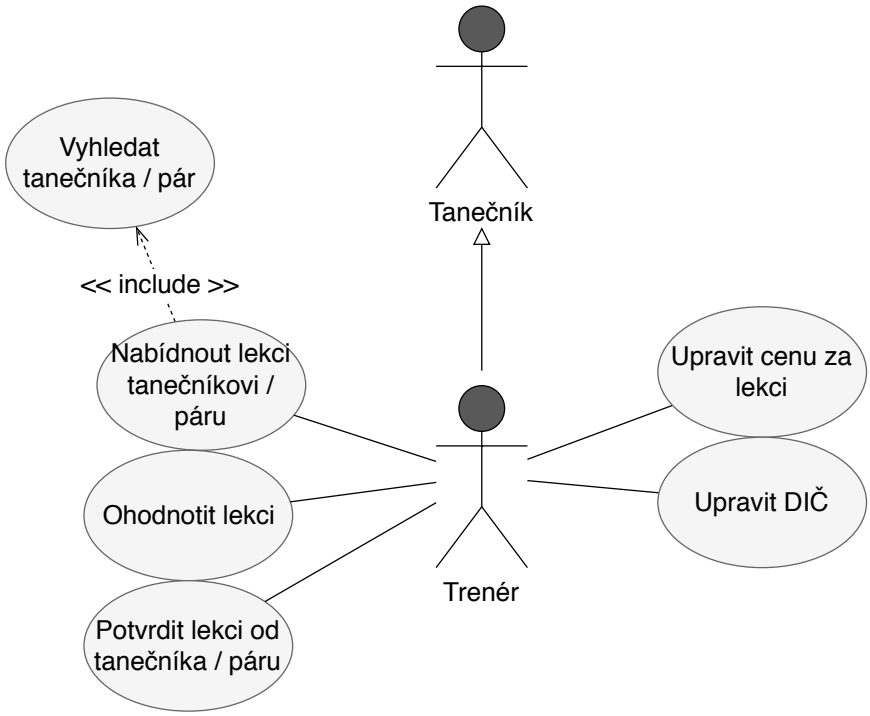
Tanečník je členem systému s nejnižším oprávněním. Tanečníkům jsou zpřístupněny zpravidla funkce zobrazení různých informací napříč systémem. Možnosti tanečníka modifikovat výrazněji stav systému jsou značně omezené. Tanečník může například upravit své osobní údaje, požádat trenéra o individuální lekci, a tu zpětně ohodnotit, případně vložit info, které vidí ostatní členové systému.



Obrázek 3.1: Diagram případů užití z pohledu tanečníka

3.7 Role trenéra

Trenér dědí všechny případy užití od tanečníka. Důvodem pro tuto volbu je fakt, že v tanečním sportu je velice běžný případ, kdy ještě aktivní tanečník je zároveň již trenérem. Role trenéra se soustředí na jednu základní oblast, a tou je správa vlastních individuálních lekcí z trenérského pohledu. Případy užití jsou v podstatě ekvivalentní s těmi u tanečníka, fungují ovšem v opačném směru (trenér nabízí tanečníkům / párům individuální lekce a hodnotí jejich přístup).



Obrázek 3.2: Diagram případů užití z pohledu trenéra

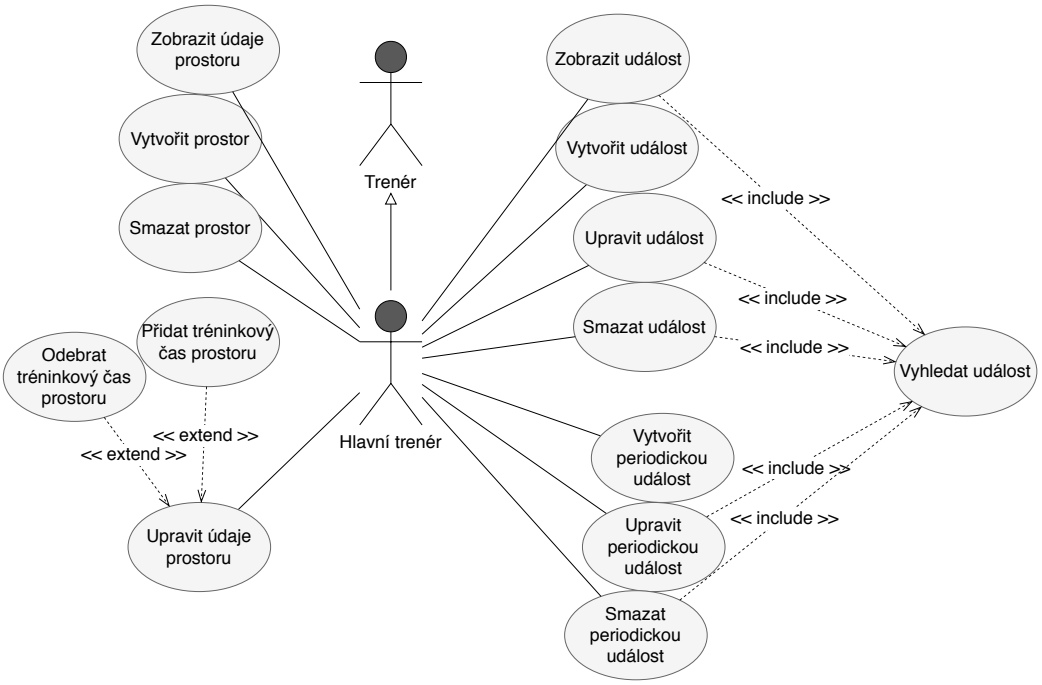
3.8 Role hlavního trenéra

Hlavní trenér má na starosti správu událostí a prostor. Samozřejmostí je, že dědí všechny případy užití od trenéra potažmo tanečníka. Hlavních trenérů se v systému může vyskytovat obecně *více*. Tato volba má tři důvody:

1. Trenéři mohou být v rámci klubu úzce svázáni s konkrétním prostorem, v těchto případech je vhodné zvolit více hlavních trenérů, aby spravovali události, které probíhají na *jejich* prostorech.

2. V případě, že je hlavní trenér po nějaký čas nedostupný, může požádat administrátora o udělení práv hlavního trenéra dalšímu (záložnímu) trenérovi.

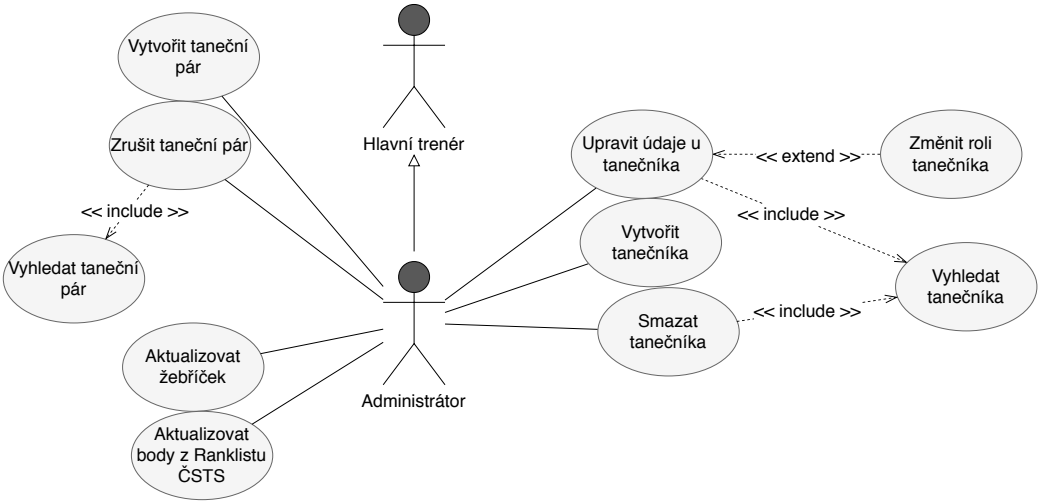
3. Ve větších klubech není realizovatelné, aby jeden hlavní trenér zvládal spravovat všechny události a prostory.



Obrázek 3.3: Diagram případů užití z pohledu hlavního trenéra

3.9 Role administrátora

Administrátor má veškerá práva manipulace se systémem. Zpravidla by měl roli administrátora zastávat jeden z hlavních trenérů. Tato role není určena administrátorovi ve smyslu externího správce celého systému, ale vůdčí osobnosti celého klubu (kromě hlavního trenéra se nabízí např. majitel klubu).



Obrázek 3.4: Diagram případů užití z pohledu administrátora

Kapitola 4

Návrh

V této kapitole představím návrh systému z pohledu databáze, uživatelského rozhraní a struktury programu.

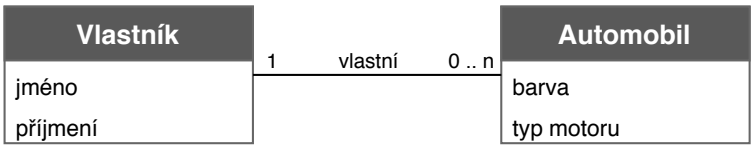
4.1 Entity-relationship model

Takzvaný entity-relationship model se používá pro abstraktní znázornění dat. Data jsou zde seskupována pro účely databáze. Diagramy, které vznikají pomocí této metody se nazývají *ER diagramy*. Prvky ER diagramu jsou:

- **Entity** - abstrakce konkrétních objektů reálného světa
- **Entitní množiny** - skupiny entit stejného typu, mají stejné *atributy*
- **Atributy** - vlastnosti entit
- **Vztahy** - asociace mezi entitami
- **Vztahové množiny** - podobně jako entitní množiny mohou mít atributy, jedná se o množinu vztahů téhož typu

4.1.1 Členství a kardinalita

Členství definuje minimální počet vztahů konkrétního typu, ve kterých musí participovat jedna entita. Kardinalita naproti tomu udává maximální počet takových vztahů.

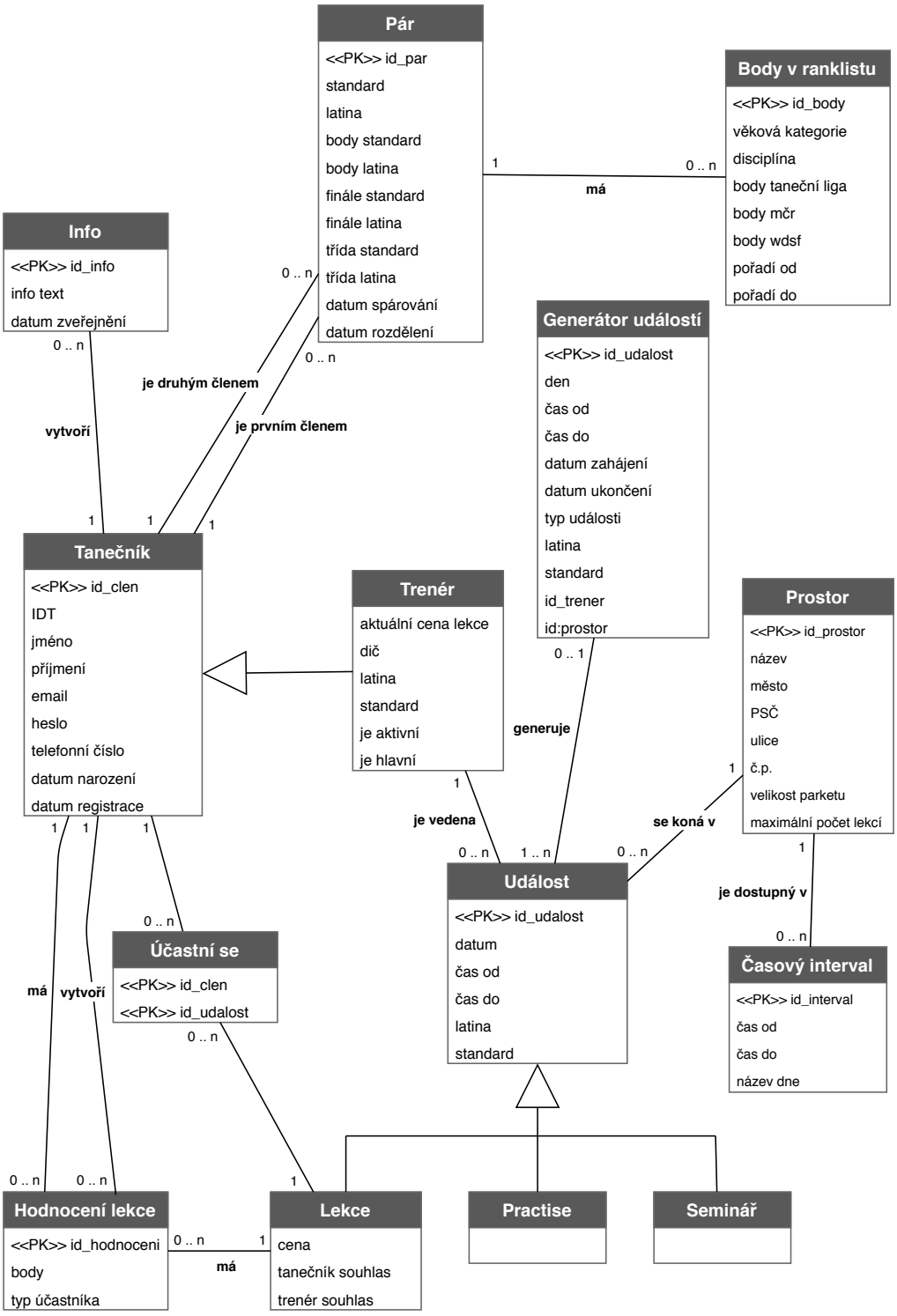


Obrázek 4.1: Jednoduchý příklad ERD

V příkladu na obrázku 4.1 můžete rozlišit dvě entitní množiny - *vlastníka* a *automobil*. Atributy vlastníka jsou *jméno* a *příjmení*, automobil má atributy *barva* a *typ motoru*. Tyto entitní množiny jsou asociovány vztahovou množinou *vlastní*. Jeden vlastník může vlastnit *0* až *n* automobilů, kdežto jeden automobil připadá právě jednomu vlastníkovi.

4.2 ER diagram systému

Na obrázku 4.2 můžete vidět ER diagram, který je výsledkem mého návrhu datové části výsledného systému.

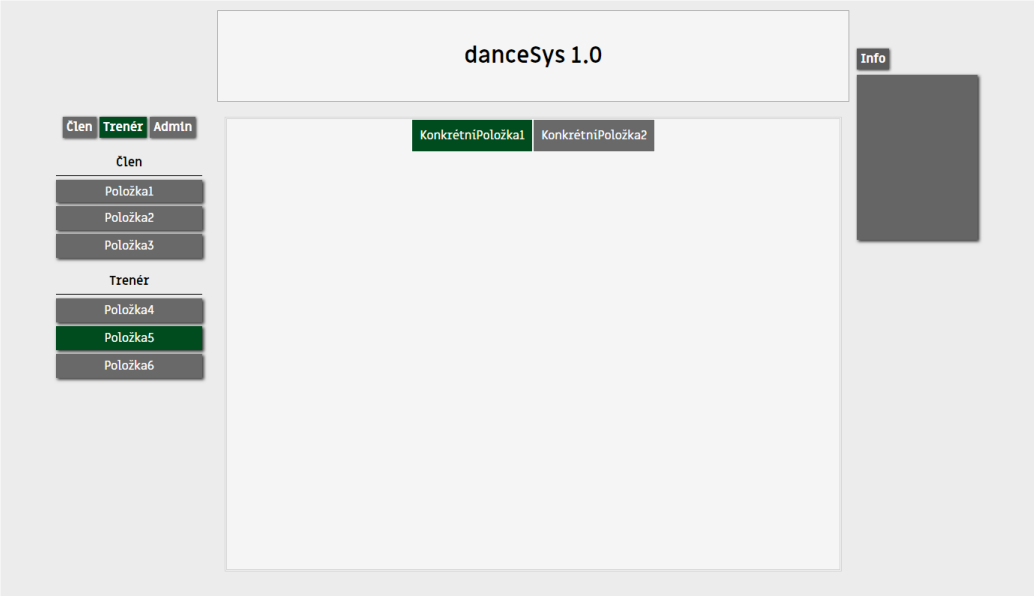


Obrázek 4.2: ER diagram IS klubu tanečního sportu

4.3 Grafické uživatelské rozhraní

Smyslem bylo navrhnout co nejjednodušší grafické uživatelské rozhraní, aby práce se systémem byla intuitivní. Uživatel by měl být schopný v co nejkratším čase a bez zdlouhavého hledání provést konkrétní úkony. Z tohoto důvodu jsem zvolil barevné oddělení funkcionality, která se týká jednotlivých rolí uživatelů. Pokud tedy uživatel zastává více rolí, systém mu pomocí konkrétního barevného zobrazení *napovídá*, pro kterou roli je daná funkcionalita určena. Konkrétně byla zvolena *tmavomodrá* barva pro tanečníky, *zelená* pro trenéry a hlavní trenéry a *červená* pro administrátorské rozhraní.

Uživatelské rozhraní nabízí jedno *hlavní menu* na levé straně obrazovky, nahoře je zobrazena *hlavička*. Hlavičku lze využít například pro *personalizaci*, lze do ní umístit logo a jméno konkrétního tanečního klubu, aby se uživatelé v systému cítili jako v domácím prostředí. Vpravo je pak zobrazeno aktuální *info*, které lze z kterékoli obrazovky také přidávat a mazat. Uprostřed se nachází *obsah* konkrétní obrazovky, někdy doplněn *kontextovým menu*, které se váže na konkrétní položku z hlavního menu. Kontextové menu je výhodné využít v případě, kdy je jedna položka hlavního menu příliš obecná na to, aby vedla na jednu konkrétní obrazovku. Například pro položku *Správa událostí* bylo vytvořeno kontextové menu s položkami *Nová událost* a *Seznam událostí*.



Obrázek 4.3: Mockup GUI systému

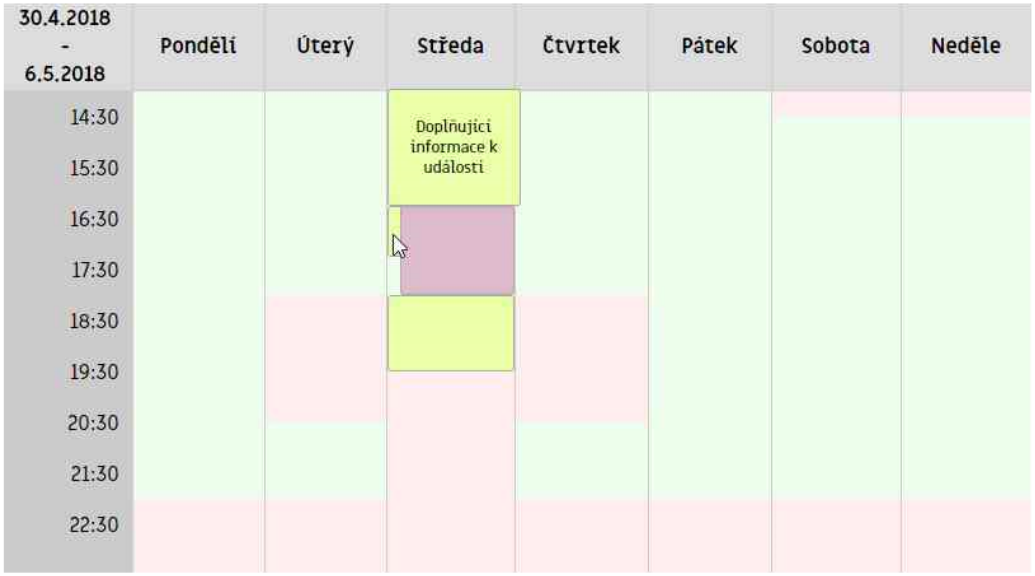
4.3.1 Rozvrh

Nejzajímavějším prvkem grafického uživatelského rozhraní výsledného systému je bezpochyby rozvrh jednotlivých prostor. Při návrhu vzhledu rozvrhu bylo třeba myslet na to, že bude obsahovat relativně velké množství informací, protože má zobrazovat všechny události konkrétního týdne v roce. Události ovšem mohou běžet *souběžně* i v rámci jednoho prostoru. Toto je běžné například při výuce individuálních lekcí.

Vznikl tedy návrh, ve kterém je konkrétní událost vyznačena v rozvrhu barvou, která odpovídá typu události. V implementaci bude rozvrh doplněn o *legendu* barev. Základní

informace jsou tedy uživateli viditelné na první pohled. Po přejetí kurzoru nad konkrétní událostí se zobrazí okénko (*tooltip*) s doplňujícími informacemi.

Zobrazování souběžně probíhajících událostí je třeba ošetřit tak, že v případě kolize budou nově vzniklé události posouvány směrem vpravo, aby nemohlo dojít ke kompletnímu překrytí některé z událostí.



Obrázek 4.4: Mockup rozvrhu prostor

Z rozvrhu lze navíc na první pohled určit také čas, kdy je prostor dostupný volnému tréninku díky zelenému zvýraznění daného časového intervalu.

Kapitola 5

Implementace

5.1 Datový model

Při implementaci datové části mého systému jsem cílil zejména na eliminaci redundantního kódu a oddělení tzv. *business logiky* od *přímého přístupu k datům* v databázi. Za tímto účelem jsem vytvořil dvě na sobě závislé vrstvy **modelů**.

5.1.1 Entity modely

Entity modely, jak již název napovídá, reprezentují konkrétní databázové tabulky. Každá z tabulek je reprezentována jedním modelem. Za účelem lepší struktury projektu jsem modely typu entity umístil do speciálního adresáře.

Při volbě tohoto typu implementace jsem byl inspirován návrhovým vzorem *Active Record*. [1]

Bázový model

Všechny entitní modely jsou potomky *bázového modelu*, který byl implementován, aby sjednotil běžné operace nad databázovou tabulkou jakými jsou např. operace *insert*, *update* nebo *getOne*. Dané metody jsem seskupil v modelu *BaseEntityModel.php*. Daný přístup má následující výhody:

- eliminace redundantního kódu
- jednotná implementace pro všechny entity

Při instanciaci bázového modelu je v tomto případě potřeba předat modelu *název tabulky* a *název primárního klíče*. V mé implementaci jsem se rozhodl, že každá tabulka bude mít výchozí atribut, podle kterého se budou záznamy řadit. Bázovému modelu tedy předávám i název tohoto atributu a směr řazení (vzestupně / sestupně).

Pro implementaci konkrétních entitních modelů tedy stačí minimum dalšího kódu, což je vidět ve fragmentu 5.1.

```php
<?php
	require_once('BaseEntityModel.php');
	class DancerEntityModel extends BaseEntityModel{

		const TABLE_NAME = 'tanecnik';
		const ID_NAME = 'id_clen';
		const DEFAULT_SORT = 'datum_registrace';
		const DEFAULT_SORT_OPT = 'ASC';

		function __construct()
		{
			parent::__construct(
				self::TABLE_NAME,
				self::ID_NAME,
				self::DEFAULT_SORT,
				self::DEFAULT_SORT_OPT
			);
		}
	}
```

Fragment kódu 5.1: Implementace entity modelu

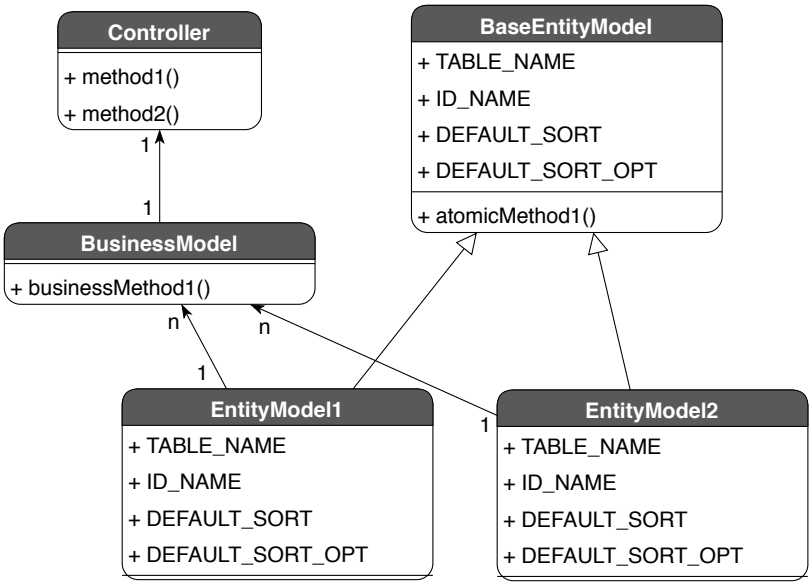
Jediné, co je třeba zajistit je inicializace bázového modelu, veškerá funkcionalita entitní třídy je implementovaná přímo v něm.

Spojení tabulek

Kromě základních operací nad *jednou* databázovou tabulkou jsem se také rozhodl implementovat v bázovém modelu možnost spojit dvě tabulky SQL klauzulí *JOIN*. V tomto případě je třeba načíst oba entitní modely, jejichž tabulky chceme spojit. Spojení je možné díky předání druhého entitního modelu jak parametr spojovací metody *BaseEntityModel::getJoined* v modelu prvním.

5.1.2 Business modely

Modely typu *business* tvoří vrstvu, která již nekomunikuje přímo s databází, ale používá metody entitních modelů. Lze tedy opakovaně využívat již napsaný kód, čímž se snižuje riziko zanesení chyb. Business modely byly vytvářeny souběžně s controllery, kterým tedy odpovídají *jedna ku jedné*. Z tohoto důvodu se v rámci business modelů nelze zcela vyhnout redundancím v kódu, což je ale malá daň za výše uvedené výhody této architektury.



Obrázek 5.1: Obecný diagram tříd controllerů a modelů

5.2 Ukládání hesel

Pokud implementujeme systém, který vyžaduje *autentizaci* uživatelů, vyvstane otázka, jakým způsobem ukládat jejich hesla do databáze. Bez delších úvah lze zavrhnout ukládání hesel v původní textové podobě. Jazyk PHP nabízí kvalitní podporu *hashování* hesel.

5.2.1 Nejčastější typy útoků

Předtím než vysvětlím mé rozhodnutí při volbě hashovacího algoritmu, rád bych krátce shrnul nejčastější typy útoků, které se snaží hesla prolomit a získat tak přístup k uživatelským účtům. [2]

Slovníkové útoky

Slovníkové útoky používají soubory, které obsahují často používaná hesla. Jednotlivé položky souboru jsou *zahashovány* a porovnány se zahashovaným uživatelským heslem. Pokud se zahashované hodnoty shodují, aktuální položka v souboru je skutečným heslem.

Na položky ze slovníků jsou často aplikovány transformační funkce, které například položku upraví záměnou *alfabetických* znaků za *numerické*, např **hello -> h3110**.

Útoky hrubou silou

Tyto útoky jsou velice prosté. Postupně se zkouší všechny možné kombinace znaků až do určité délky. Pro zvýšení ochrany před těmito útoky je vhodné specifikovat *minimální délku hesla* a *povinnost* uživatele zahrnout do hesla znaky z různých množin (malá, velká písmena, číslice, ...). Těmito metodami lze enormně zvýšit výpočetní náročnost tohoto typu útoku.

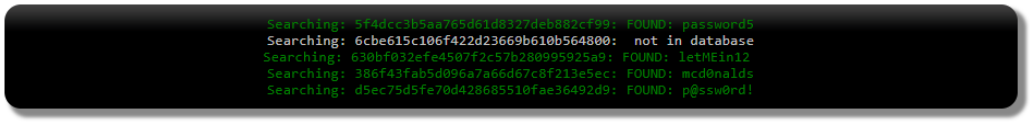
Proti slovníkovým útokům a útokům hrubou silou neexistuje stoprocentní ochrana. Pokud je náš systém hashování hesel bezpečný, měly by být výše uvedené typy útoků jediné, kterými lze heslo odhalit.

Obrázek 5.2: Příklad slovníkového útoku a útoku hrubou silou [2]

Použití vyhledávací tabulky

Ve vyhledávací tabulce jsou zahashovaná hesla (obvykle jsou použita hesla z dostupných slovníků hesel) a jejich odpovídající hesla v prosté formě. Vstupem této metody je hash skutečného uživatelského hesla, který je porovnáván s jednotlivými záznamy ve vyhledávací tabulce, dokud není nalezena shoda. Jedná se tedy o částečnou implementaci *reverzní funkce*, která jinak v oblasti hashování neexistuje.

Metoda vyhledávací tabulky je výrazně efektivnější než předchozí metody. Její nevýhodou jsou ovšem velké nároky na paměťový prostor. Kompromis mezi rychlostí odhalení hesla a nároky na pamět nabízí *duhové tabulky*.



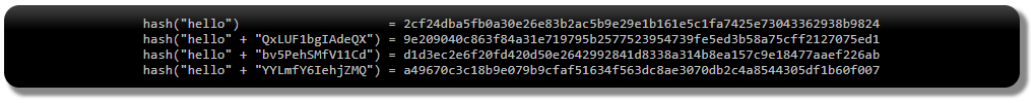
Obrázek 5.3: Příklad útoku s vyhledávací tabulkou [2]

5.2.2 Kryptografická sůl

Použití hashovacího algoritmu za přítomnosti kryptografické soli lze zcela eliminovat útoky, které používají *vyhledávací tabulky*.

Sůl je náhodný řetězec, který je *konkatenován* s heslem ještě před spuštěním hashovacího algoritmu. Tímto je zajištěno, že každá instance hesla má různý hash i v případě, kdy jsou původní hesla shodná. Hodnotu soli je nutné uchovávat (např. v databázi), aby mohlo dojít k ověření hesla.

Útoky za pomoci vyhledávací tabulky nejsou v tomto případě realizovatelné, ta totiž funguje pouze tehdy, je-li konkrétní heslo zahashováno vždy do stejné podoby.



Obrázek 5.4: Přidání kryptografické soli [2]

5.2.3 Volba implementace

V mé implementaci hashuji hesla pomocí vestavěné PHP funkce *password_hash*. Ta umožňuje volbu implementace hashovací funkce. Zvolil jsem tedy funkci **bcrypt**, která převádí vstupní řetězec na posloupnost 60 znaků, která obsahuje:

- prefix **$2y$**, který udává, že se jedná o modulární šifrovací formát
- parametr *cost* např. **10$**, který specifikuje, jaká časová složitost algoritmu se má použít (čím větší, tím lepší ochrana před útoky hrubou silou)
- 22 znaků **kryptografické soli** v base64 kódování
- 31 znaků **zahashovaného hesla** v base64 kódování

Hesla jsou tedy chráněna před útoky za použití vyhledávací tabulky díky tomu, že zvolená implementace podporuje hashování s kryptografickou solí. Prosté útoky pomocí slovníku nebo útoky hrubou silou jsou znesnadněny vhodně zvolenou časovou složitostí.

5.3 Generování rozvrhu

Jak bylo uvedeno v kapitole s návrhem systému, systém podporuje generování rozvrhu jednotlivých prostor, tedy jakýsi způsob vizualizace událostí a tréninkových časů, který je pro zobrazení uživatelům vhodnější než prosté seznamy.

Při implementaci generátoru nebylo využito žádné *externí* knihovny. Byly použity pouze prostředky standardní knihovny jazyka PHP a vlastní pomocné třídy.

5.3.1 Třída DOMDocument

DOMDocument slouží pro reprezentaci *HTML* či *XML* dokumentu. Lze v něm vytvořit kompletní stromovou strukturu a uložit ji například do souboru. Třída podporuje všechny očekávané operace nad dokumentem jako je vytváření elementů, přidávání referencí na potomky, nastavování atributů a také vkládání textových dat.

5.3.2 Třída ScheduleGenerator

Jedná se o vlastní implementaci generátoru. *Jedna* instance této třídy reprezentuje *jeden* generovaný týden v *konkrétním* prostoru. Pseudokód pro vygenerování všech týdenních rozvrhů je znázorněn ve fragmentu 5.2.

```
<?php
        foreach($places as $place){
                foreach($weeksGenerated as $week){
                        $events = getEvents($place, $week);
                        $scheduleGenerator = new ScheduleGenerator();
                        $scheduleGenerator->setData($events, $place, $week);
                        $scheduleGenerator->generate();
                }
        }
```

Fragment kódu 5.2: Pseudokód pro generování týdenních rozvrhů

ScheduleGenerator používá přímo metody DOMDocumentu, za jejichž pomocí vytváří konkrétní elementy, které odpovídají událostem v databázi. Nosná logika rozvrhu se odehrává v elementu třídy *dayWrapper*, což je buňka tabulky. Touto buňkou je reprezentován jeden den v týdnu. Při umisťování událostí do rozvrhu je třeba brát v potaz tyto skutečnosti.

Počáteční čas události

Počáteční čas události slouží k určení *offsetu* od horního okraje buňky třídy *dayWrapper*. Offset je dále pronásoben třídní konstantou, probíhá tedy mapování jednotkového offsetu na konkrétní počet pixelů, které jsou předány CSS stylům konkrétní události.

Koncový čas události

Rozdíl koncového a počátečního času nám udává *délku události*. Ta je následně zpracována stejným způsobem jako offset od horního okraje. Společná násobící konstanta nám zajistí konzistenci mezi offsety a délkami událostí.

Dosud vytvořené události

V rámci jedné buňky třídy *dayWrapper* je kromě offsetu události od horního okraje důležité stanovit offset od levého okraje buňky. Vzniká tedy nutnost ukládat již vytvořené události a kontrolovat časové kolize. K názornějšímu popisu, jak se stanoví offset od levého okraje jsem zvolil popis v pseudokódu ve fragmentu 5.3.

```
<?php
        $alreadyCreatedEvents = array();
        foreach($eventsInDay as $event){
                $offset = 0;
                foreach($alreadyCreatedEvents as $alreadyCreatedEvent){
                        if($event->collide($alreadyCreatedEvent)){
                                if($alreadyCreatedEvent->offset >= $offset){
                                        $offset = $alreadyCreatedEvent->offset + 1;
                                }
                        }
                }
                $array_push($alreadyCreatedEvents, $event);
        }
```

Fragment kódu 5.3: Pseudokód pro kontrolu kolizí událostí

Typ události

Podle typu události je nastavena barva pozadí události a také text, který se zobrazí po přejetí kurzoru přes událost.

5.3.3 Pravidelná aktualizace

Není vhodné generovat rozvrh při každém přístupu k obrazovce s rozvrhem. Proto vedle souborů s jednotlivými týdenními rozvrhy vzniká také po každém vygenerování soubor *timeStamp.txt*, kam se ukládá datum a čas, kdy proběhlo poslední vygenerování. Pokud

rozdíl mezi aktuálním časem a hodnotou v souboru přesáhne stanovený limit, vyvolá se s následujícím přístupem k obrazovce rozvrhu také proces generování.

Pokud uživatel potřebuje vidět aktuální rozvrh ihned, lze aktualizaci *vyvolat* explicitně přímo na obrazovce s rozvrhem.

5.4 Periodické události

Součástí systému pro hlavní trenéry je možnost vytvářet tzv. *periodické události*. Ty slouží k vygenerování několika instancí konkrétních událostí tak, aby uživatel nemusel vytvářet stejnou událost pro každý týden zvlášť.

5.4.1 Vytvoření periodické události

Vytvoření periodické události je možné na obrazovce *Nová událost* v sekci *Správa událostí*. Kromě atributů, které jsou třeba k vytvoření běžné události je třeba zadat také **počáteční** a **koncové datum**, aby se systém nepokusil vygenerovat nekonečně mnoho událostí.

Vyplněný formulář je zpracován v *Eventmanager* controlleru funkcí *createPeriodicalEvent*, která vytvoří:

1. Objekt *generator* připravený k importu do databáze, který drží vlastnosti, společné pro každou událost.
2. Pole objektů *events*, které reprezentuje jednotlivé události připravené k importu do databáze.

Vznik konkrétních událostí

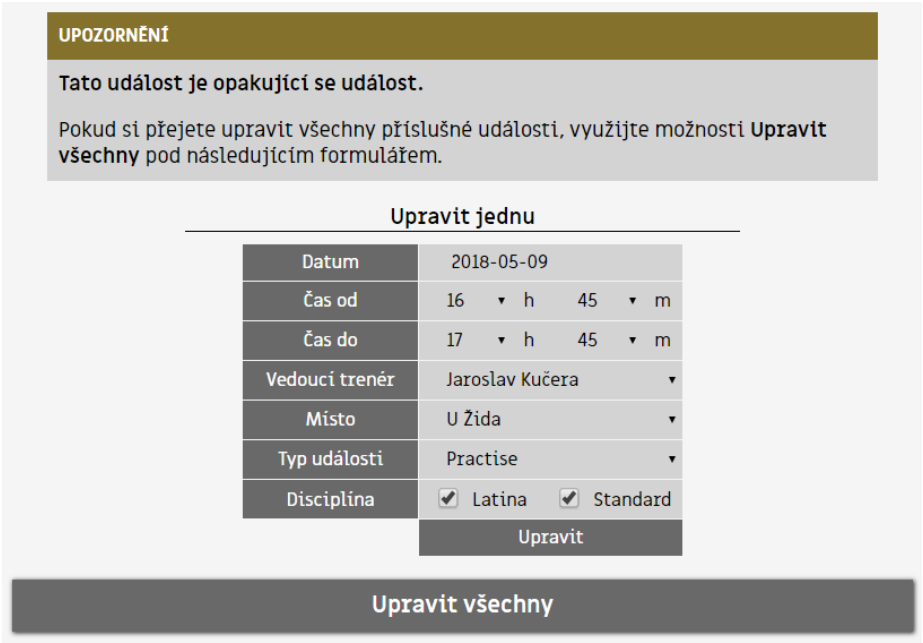
Generování konkrétních událostí je odstíněno od samotné logiky controlleru. Připravený generátor je předán funkci *Globalfunctions::createEventsFromGenerator*, kde dochází ke:

- konkretizaci data události
- překopírování hodnot z generátoru do konkrétní události
- nastavení odkazu (cizího klíče) z události na generátor

5.4.2 Úprava periodické události

Vzhledem k tomu, že v databázi vznikne jak záznam generátoru, tak i jednotlivé záznamy konkrétních událostí, lze dát uživateli prostor upravit nejen jednu z vygenerovaných událostí, ale i všechny vygenerované události. To je možné díky referenci mezi událostí a generátorem. Pokud uživatel upravuje události, ke které existuje v databázi generátor, je na tuto skutečnost v obrazovce s úpravou události upozorněn.

Ovlivněny jsou pouze události, které k danému dni ještě neproběhly.



Obrázek 5.5: Upozornění pro uživatele upravující periodickou událost

5.4.3 Mazání periodické události

Události lze mazat jak jednotlivě, tak všechny najednou. Pokud se tedy uživatel rozhodne smazat některou z událostí, která byla vygenerována generátorem, zobrazí se mu podobné upozornění jako na obrazovce s úpravou události.

Obrázek 5.6: Upozornění pro uživatele na obrazovce mazání periodické události

5.5 Stahování dat z ČSTS

V další části implementace bylo třeba vyřešit aktualizaci dat v žebříčku párů. Spuštění aktualizace probíhá na vyžádání uživatele, který musí disponovat administrátorským oprávněním.

5.5.1 Předpoklady

Pro umožnění automatické aktualizace je třeba evidovat u tanečníků našeho systému jejich **IDT**, které jim bylo přiřazeno svazem. Na základě toho lze tanečníka v systému svazu vyhledat.

V potaz jsou brány pouze páry, které jsou spárované v našem systému a řadí se mezi *aktivní páry* - mají atribut *datum_rozdeleni* nastavený na **NULL**.

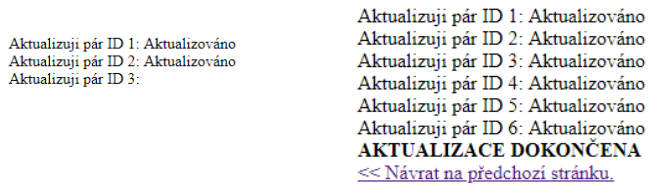
5.5.2 Aktualizace žebříčku postupových soutěží

Pro aktualizaci dat z postupových soutěží je třeba vzít IDT jednoho tanečníka z daného páru a předat jej funkci *Scrapper::getDataByIdt*. Třída *Scrapper* zapouzdřuje logiku získání dat jednoho tanečníka. Zmíněná funkce provádí postupně následující kroky:

1. Z **IDT** vytvoří adresu URL, která vede na stránku konkrétního tanečníka.
2. Uloží si HTML obsah stránky do stromové struktury.
3. Vyhledá konkrétní elementy a uloží jejich textový obsah do vytvořeného objektu.
4. Vrátí objekt naplněný daty.

Objekt s konkrétními hodnotami následně slouží k aktualizaci jednoho páru v databázi.

Vzhledem k tomu, že aktualizace dat u většího počtu párů může zabrat i desítky vteřin, je uživatel průběžně informován o stavu aktualizace (obrázek 5.7).



Obrázek 5.7: Informace o průběhu aktualizace

Simple HTML DOM Parser

Kvůli zefektivnění vyhledávání jednotlivých HTML elementů na stránce tanečníka byla použita externí knihovna *Simple HTML DOM Parser*. Tato knihovna umožňuje načíst celý dokument HTML a následně v něm vyhledávat pomocí selektorů, jejichž zápis je podobný zápisu CSS selektorů, viz fragment 5.4.

```php
<?php
        $url = 'http://www.csts.cz/cs/Clenove/Detail/'.$idt;
        $html = simplehtmldom_1_5\file_get_html($url);
        $table = $html->find('table.tab-simple2');
        $tableText = $table->text();
```

Fragment kódu 5.4: Ukázka použití knihovny Simple HTML DOM Parser

Vizualizace žebříčku

Při přístupu do databáze jsou páry řazeny podle speciálních pravidel tak, aby jejich pořadí v žebříčku odpovídalo jejich výkonnostní kategorii, počtům finálových umístění a získaným bodům.

Obrázek 5.8: Zobrazení žebříčku postupových soutěží

5.5.3 Aktualizace bodů z Ranklistu ČR

S Ranklistem ČR byla práce daleko přímočařejší než v případě postupových soutěží. Webové stránky svazu totiž umožňují dotazovat se na Ranklist ČR prostřednictvím *API*. API bylo nutné si lehce nastudovat, abych zjistil, v jakém formátu posílat dotazy a naopak, jak interpretovat data, která mi přijdou v odpovědi od serveru.

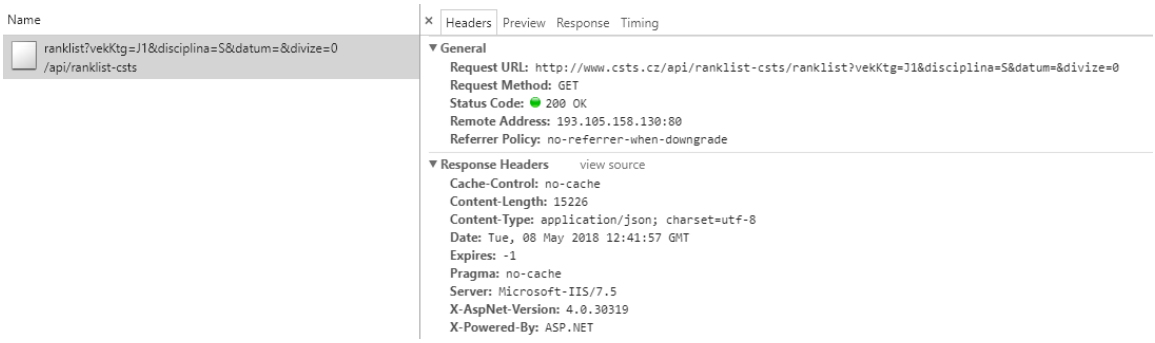
API

Application programming interface je rozhraní programu, které umožňuje interakci s jiným softwarem, podobně jako *uživatelské rozhraní* umožňuje interakci s uživatelem. Častým využitím API ve světě webových aplikací je umožnit jiným aplikacím získat určitou podmnožinu vlastních dat v jednotném formátu. Taková data se pak dají velice snadno dále zpracovávat.

Znamená to také, že i když se změní uživatelské rozhraní aplikace, API může poskytovat data dále v původním formátu, není třeba měnit implementaci získání aplikačních dat.

Práce s API svazu

Při průzkumu API mi velice pomohlo vývojářské prostředí prohlížeče Google Chrome. Prostředí je ukázáno na obrázku 5.9.



Obrázek 5.9: Vývojářské prostředí - HTTP dotaz na API

Z obrázku 5.9 lze vyčíst dvě důležité informace.

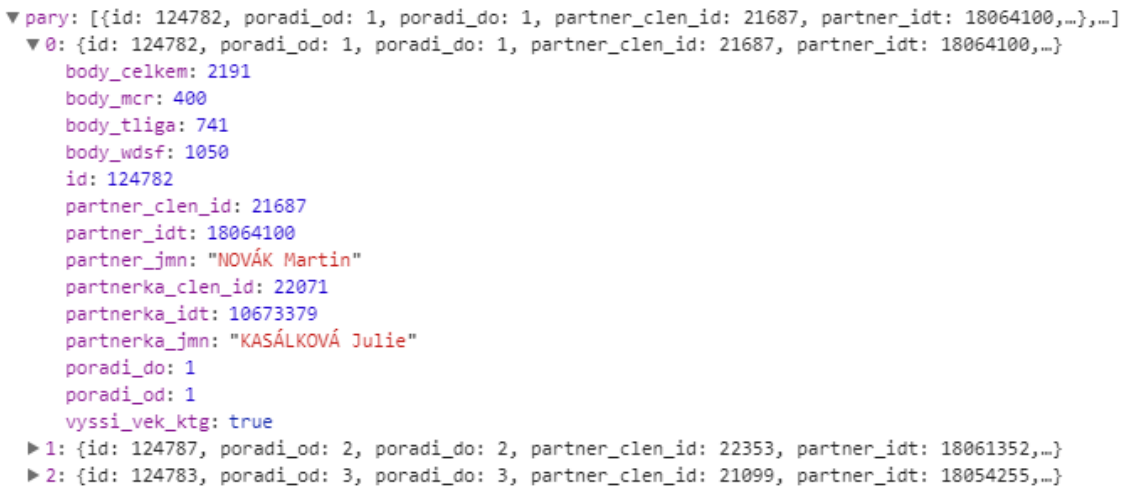
1. dotazování na API probíhá prostřednictvím HTTP metody **GET**, parametry dotazu se tedy předávají na konci URL (část za otazníkem)
2. API nám vrátí data ve formátu **JSON**

Následně jsem přešel k prozkoumání struktury přijaté odpovědi.


```
▶ rank: {id: 800, datum: "2018-05-01T00:00:00", prubezny: true, profi: false, vekKtg: "J1", disciplina: "S",…}
▼ selection: {datum: "", vekKtg: "J1", disciplina: "S", divize: 0,…}
    datum: ""
    disciplina: "S"
    divize: 0
  ▶ seznamDatumu: [{Key: "", Value: "průběžný"}, {Key: "2018-04-01", Value: "2018-04-01"},…]
  ▼ seznamDisciplin: [{Key: "S", Value: "Standard"}, {Key: "L", Value: "Latina"}, {Key: "K", Value: "10 tanců"}]
    ▶ 0: {Key: "S", Value: "Standard"}
    ▶ 1: {Key: "L", Value: "Latina"}
    ▶ 2: {Key: "K", Value: "10 tanců"}
  ▶ seznamDivizi: [{Key: 0, Value: "- všechny -"}, {Key: 1, Value: "Pražská divize"},…]
  ▼ seznamVekovychKategorii: [{Key: "J1", Value: "Junioři I."}, {Key: "J2", Value: "Junioři II."}, {Key: "ML", Value:
    ▶ 0: {Key: "J1", Value: "Junioři I."}
    ▶ 1: {Key: "J2", Value: "Junioři II."}
    ▶ 2: {Key: "ML", Value: "Mládež"}
    ▶ 3: {Key: "21", Value: "Do 21 let"}
    ▶ 4: {Key: "DS", Value: "Dospělí"}
    ▶ 5: {Key: "PD", Value: "PD - Profesionálové"}
    ▶ 6: {Key: "S1", Value: "Senioři I."}
    ▶ 7: {Key: "S2", Value: "Senioři II."}
    ▶ 8: {Key: "S3", Value: "Senioři III."}
    ▶ 9: {Key: "S4", Value: "Senioři IV."}
    vekKtg: "J1"
```


Obrázek 5.10: Struktura JSON odpovědi

Z odpovědi na obrázku 5.10 lze snadno zjistit, jak jsou v rámci API pojmenovány jednotlivé *disciplíny* a *věkové kategorie*. Uzly JSON objektu *seznamDatumu* a *seznamDivizi* nebude třeba zpracovávat, jelikož chceme získávat vždy průběžné hodnoty. Specifikováním divize by se sice dotazování značně zrychlilo, ale zároveň neúměrně více zkomplikovalo, protože by bylo potřeba vyřešit vyhledávání tanečníků, kteří jsou v klubu na hostování z divize jiné.



Obrázek 5.11: Struktura JSON odpovědi - konkrétní pár

Na obrázku 5.11 lze vidět, že jsou páry uloženy jako *kolekce struktur*, ve které každá struktura již obsahuje konkrétní datové položky, ve kterých jsou potřebné hodnoty. Pro účely systému bylo třeba brát v potaz následující položky:

- *body_mcr*, *body_tliga* a *body_wdsf* pro získání konkrétního počtu bodů daného páru
- *partner_idt* a *partnerka_idt* pro identifikaci páru
- *poradi_od* a *poradi_do* pro určení pořadí páru

Konkrétní implementace

Pro stažená data jsem vyhradil novou databázovou tabulku **body_ranklist**, která odkazuje na tabulku **par**. Aktualizace dat z Ranklistu ČR se dá shrnout do tří hlavních kroků.

1. Získání čísel IDT aktivních páru z vlastní databáze.
2. Vyprázdnění (*truncate*) tabulky **body_ranklist**.
3. Dotázání se na API, iterace nad výsledky, případné vložení záznamu do tabulky **body_ranklist**.

Třetí krok probíhá opakovaně pro jednotlivé disciplíny a věkové kategorie. Nejprve je potřeba odeslat *úvodní dotaz*, kterým získáme seznam disciplín a věkových kategorií (fragment 5.5).

```php
<?php
        $json = file_get_contents('http://www.csts.cz/api/ranklist-csts/ranklist');
        $decoded = json_decode($json);
        $categories = $decoded->selection->seznamVekovychKategorii;
        $disciplines = $decoded->selection->seznamDisciplin;
```

Fragment kódu 5.5: Provedení úvodního dorazu na API

Následně iterujeme nad disciplínami a věkovými kategoriemi. V příchozích odpovědích hledáme shodu v IDT obou partnerů (fragment 5.6).

```php
<?php

foreach($categories as $category){
        $c = $category->Key;
        foreach($disciplines as $discipline){
                $d = $discipline->Key;
                $json = file_get_contents($baseUrl.'?vekKtg='.$c.'&disciplina='.$d);
                $decoded = json_decode($json);
                foreach($decoded->rank->pary as $couple){
                        foreach($localCouples as $localCouple){
                                /* Code for saving points to database
                                 * if couples' IDTs match.
                                 */
                        }
                }
        }
}
```

Fragment kódu 5.6: Implementace získání bodů konkrétních párů

Pár se může účastnit soutěží v různých disciplínách, v některých případech také v různých věkových kategoriích (většinou je běžné, že mladší páry tancují také ve starší kategorii). Jeden pár tedy může být během iterování nalezen až devětkrát. Proto není možná optimalizace algoritmu tím, že bychom z lokálního pole párů odstraňovali již nalezené páry.

Vizualizace bodů

Body z Ranklistu ČR je možné zobrazit přímo na obrazovce žebříčku postupových soutěží, a to po kliknutí na tlačítko **RL ČR**. Toto tlačítko je vykreslováno pouze u párů, které se v Ranklistu ČR vyskytují.

Obrázek 5.12: Zobrazení bodů z Ranklistu ČR

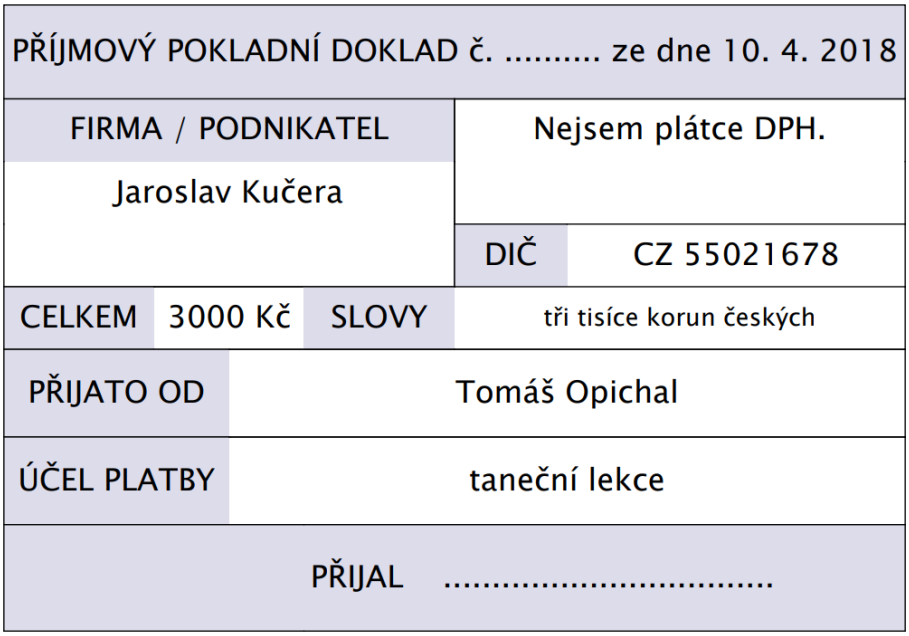
5.6 Generování dokladů za lekce

Pro účely generování dokladů ve formátu PDF byla použita již existující třída **tFPDF**. Ta je rozšířením třídy FPDF, do které přináší navíc podporu kódování UTF-8. Jako mezičlánek

pro práci s touto knihovnou jsem vytvořil knihovní třídu **ReceiptCreator**. Kroky, které se při generování dějí na pozadí jsou:

1. Získat data o konkrétní lekci z databáze.
2. Vytvořit instanci třídy ReceiptCreator a předat jí data.
3. Zavolat metodu *ReceiptCreator::createCashReceipt*, která vytvoří z dostupných dat *tFPDF* objekt.
4. Zavolat metodu *ReceiptCreator::getCashReceipt*, která zobrazí uživateli hotové PDF.

Podmínkou pro úspěšné vygenerování dokladu je platná hodnota DIČ, kterou musí mít v databázi nastavenou trenér, který lekci vedl. V opačném případě generování skončí neúspěchem s odpovídající chybovou hláškou.



Obrázek 5.13: Ukázka vygenerovaného příjmového dokladu

5.6.1 Cena slovy

Vzhledem k tomu, že se na dokladech tohoto typu uvádí také cena slovy, bylo třeba tuto funkcionalitu implementovat také. Původní záměr bylo ukládat cenu slovy jako další atribut trenéra. Tato volba by však znemožnila generování dokladů za více lekcí najednou. Pak totiž cena odpovídá násobku ceny lekce jedné a uložený řetězec tedy ztrácí význam.

Proto byla implementována vlastní funkce *GlobalfFunctions::getPriceString*, která dokáže transformovat číselnou hodnotu na cenu slovy.

- vstupem je **číselná hodnota** v rozsahu 0 až 9999
- výstupem je řetězec odpovídající ceně slovy dané vstupní hodnoty

Rozsah povolených vstupních hodnot byl zvolen na základě vlastních zkušeností s cenami individuálních lekcí z praxe.

Funkce *GlobalfFunctions::getPriceString* zpracovává vstupní hodnotu od nejvyššího řádu (tisíce). Algoritmus, který se opakuje i pro nižší řády, funguje pro tisíce následovně:

1. Celočíselným dělením číslem 1000 zjistím počet tisíců.
2. Počet tisíců převedu na řetězec pomocí definovaného asociativního pole.
3. Od původní hodnoty odečtu zjištěný počet tisíců a pokračuji na nižší řád.

Vzhledem k tomu, že výsledný řetězec může mít různou *délku*, je délka řetězce kontrolována a je jí přizpůsobena velikost *fontu* při samotném generování.

5.7 Systém hodnocení lekcí

Vzhledem k požadavkům na systém byl implementován systém *anonymního* hodnocení proběhlých individuálních lekcí. Systém umožňuje hodnotit jak trenéry, tak tanečníky. Hodnocení probíhá na obrazovce **Hodnocení** v sekci **Správa lekcí**. Na stejné obrazovce je také zobrazeno vlastní hodnocení. Jedná se o bodový systém, lze tedy hodnotit hodnotami *jedna* až *deset*, kdy deset bodů znamená *nejlepší* hodnocení. Faktory, které by měly udávat výsledné hodnocení jsou popsány v sekci 3.4.

5.7.1 Vlastní hodnocení

Vlastní hodnocení sestává ze dvou následujících hodnot:

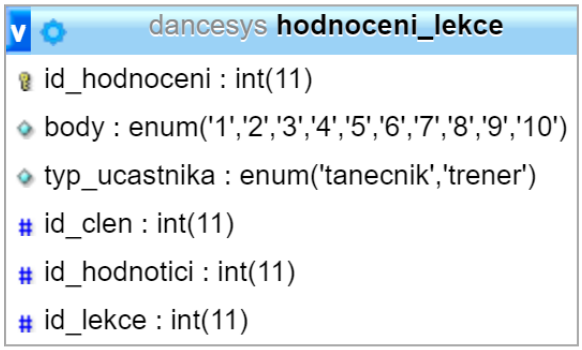
- průměrné hodnocení za minulý měsíc
- celkové průměrné hodnocení (bez hodnocení aktuálního měsíce)

Není tedy možné zobrazit jednotlivá hodnocení. Celkové hodnocení se navíc neaktualizuje ihned po vložení nového hodnocení, ale vždy první den začínajícího měsíce, kdy je do průměru započítán další měsíc.

Porovnáváním celkového průměru a průměru za poslední měsíc lze získat požadovanou zpětnou vazbu. Členové mohou zjistit, že je jejich přístup k lekcím hodnocen za poslední měsíc jinak než dříve a svůj přístup podle toho upravit.

5.7.2 Detail implementace

Implementaci vysvětlím popsáním databázové tabulky **hodnoceni_lekce** (obrázek 5.14).



Obrázek 5.14: Databázová tabulka hodnoceni_lekce

- Atribut **id_clen** je cizím klíčem do tabulky *tanecnik*, ukazuje na tanečníka nebo trenéra, kterému je hodnocení přiděleno.
- Atribut **typ_ucastnika** udává, zda se hodnocení týká členovy taneční nebo trenérské sekce.

Význam ostatních atributů je zřejmý z jejich názvu. Uvedený způsob implementace databázové tabulky umožňuje oboustrannou kontrolu, zda již hodnocení bylo uděleno. Pokud trenér vede lekci s tanečním párem, je mu umožněno ohodnotit oba členy páru zvlášť, stejně jako trenér může být ohodnocen oběma členy páru. Pokud není lekce ohodnocena do jednoho týdne od jejího konání, účastníci ztrácí nárok na provedení hodnocení.

Kapitola 6

Testování

Abych byl schopen lépe odhalit implementační chyby a nedostatky systému, zařadil jsem do konečné fáze vývoje dva druhy testů.

6.1 Testování implementace

Po implementaci každé nové funkcionality došlo na její otestování včetně otestování jejího vlivu na jiné části systému, jak uvedu na následujícím příkladu.

1. Bylo implementováno *vytváření periodických událostí*.
2. Několik takových událostí bylo vytvořeno, události byly správně uloženy v databázi a zobrazeny v seznamu událostí.
3. Byl vygenerován nový rozvrh.
4. Nové události způsobily, že se některé události v rozvrhu překrývaly takovým způsobem, že nebyly viditelné.
5. Byla zjištěna chyba v implementaci metody, která kontroluje kolize v rozvrhu.
6. Chyba byla opravena, nový rozvrh byl vygenerován správně.

Po dokončení implementace byl systém otestován na větším množství dat, které by mohlo odpovídat reálnému chodu takového systému.

Systém byl také nahrán do nového prostředí, kterým byl fakultní server *eva*. Tento krok pomohl lépe určit, které kroky se musí vykonat pro správnou konfiguraci systému v novém prostředí. Vzhledem k tomu, že na serveru eva běží operační systém, který je *case-sensitive* (rozlišuje mezi *uppercase* a *lowercase* symboly), pomohl mi odhalit několik chyb v *názvech souborů*.

6.2 Testování uživatelského rozhraní

Ve finální fázi implementace jsem systém zpřístupnil také jiným uživatelům, abych zjistil, zda je uživatelské rozhraní dostatečně intuitivní a zda se uživatelům se systémem dobře pracuje. Zaměřil jsem se na dvě skupiny uživatelů, a to *tanečníky* a *kolegy z fakulty*.

6.2.1 Názory tanečníků

Tanečníci si zejména pochvalovali *rozvrh*, kde je zaujal nápad s barevným rozlišením a zobrazení dodatečných informací po přejetí nad událostí. Další pozitivní ohlasy sklidil *info kanál*, zejména pak vytváření infa z kterékoli obrazovky systému. Co se týče správy událostí, ocenili především vyhledávací filtr.

Diskuzi částečně otevřela obrazovka s rozvrhem, která v případě, že se v systému nenachází žádný konkrétní prostor, nezobrazuje žádné informace. Někteří uživatelé byli částečně zmateni touto skutečností, proto jsem se na tuto obrazovku rozhodl přidat varování, které se zobrazí, pokud systém žádné prostory neobsahuje.

6.2.2 Názory kolegů

Kolegové z fakulty si, stejně jako tanečníci, pochvalovali možnost filtrování událostí v seznamu událostí. Dále ocenili jednotný vzhled jednotlivých obrazovek systému včetně jednotného stylu pro zobrazení seznamů.

Došlo k úpravě info kanálu, nadpis **Info** původně působil dojmem, že na něj lze kliknout. Na základě zpětné vazby také došlo k lokalizaci chybových hlášek při zpracování formulářů do češtiny, kterou sice *CodeIgniter* implicitně nepodporuje, ale implementace jazykových balíčků se v rámci MIT licence dají získat.

Kapitola 7

Závěr

Informační systém úspěšně prošel všemi fázemi vývoje, které jsou popsány v předchozích kapitolách. Všechny požadavky na systém byly implementovány s využitím vlastních schopností a za pomoci některých již existujících řešení.

Jádro systému tvoří evidence událostí, které probíhají na jednotlivých prostorech. Speciálními událostmi jsou individuální lekce, možnost jejich vytváření přímo v systému zjednodušuje proces plánování těchto lekcí oběma zúčastněným stranám. Lze také generovat automaticky generované příjmové doklady k odpovídajícím lekcím ve formátu PDF. Systém umožňuje dát účastníkům lekce anonymní zpětnou vazbu formou *bodového hodnocení*. Dále byl implementován rozvrh událostí, kde jsou události zobrazovány všem uživatelům systému. Součástí systému je evidence *tanečních párů*, jejichž body ze soutěží jsou automaticky stahovány z webu http://www.csts.cz na vyžádání administrátora.

Systém v jeho současné podobě je připraven na nasazení, zároveň je zde však velký prostor pro možné rozšíření systému.

7.1 Další vývoj

Implementovaný informační systém bude možné v budoucnu rozšířit o další dílčí moduly. Prostorem pro vylepšení je například evidence individuálních lekcí, která by mohla sloužit jako základ pro vybudování účetního systému tanečního klubu. Dále lze uvažovat o modulu, který by umožňoval vytvářet události typu *taneční soustředění*, která trvají i několik dní. V tomto modulu by mohl být použit již existující rozvrh (pro rozvrh soustředění), vytváření událostí v rámci konkrétního soustředění apod.

Další cestou by mohla být větší spolupráce s tanečním svazem, například napojení na databázi pořádaných soutěží a možnost spravovat své soutěže (přihlašování, odhlašování, zobrazení výsledků, ...) přímo v systému tanečního klubu. Dále také vylepšení stahování dat, která se týkají postupových soutěži, kde je zatím nutné pro získání dat zpracovávat výsledný HTML dokument.

V neposlední řadě lze také uvažovat o vzniku mobilní aplikace, které by sloužila jako ekvivalent informačního systému.

Literatura

[1] Beschi, J.: *Active Record Design pattern*. [Online; navštíveno 02.05.2018].
URL http://www.jacopobeschi.com/post/active-record-design-pattern

[2] Defuse Security: *Salted Password Hashing - Doing it Right*. [Online; navštíveno 03.05.2018].
URL https://crackstation.net/hashing-security.htm

[3] Dowling, T.: *Working with HTML, CSS, and HTTP*. [Online; navštíveno 28.04.2018].
URL https://journals.ala.org/index.php/ltr/article/view/4472

[4] Gilmore, W. J.: *Velká kniha PHP5 a MySQL*. Zoner Press, 2005, ISBN 80-86815-20-X.

[5] Jaroslav Zendulka, I. R.: *Databázové systémy IDS*. FIT VUT v Brně, 2006.

[6] Manuel Lemos: *4 Reasons Why All PHP Frameworks Suck?* [Online; navštíveno 29.04.2018].
URL https://www.phpclasses.org/blog/post/226-4-Reasons-Why-All-PHP-Frameworks-Suck.html

[7] Rathi, B.: *A Comprehensive Look at jQuery Selectors*. [Online; navštíveno 30.04.2018].
URL https://www.sitepoint.com/comprehensive-jquery-selectors

[8] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: *Užitečné rady pro psaní odborného textu*. FIT VUT v Brně, Listopad 2008, [Online; navštíveno 12.05.2015].
URL http://www.fit.vutbr.cz/info/statnice/psani_textu.html

[9] Stian Reimers, N. S.: *A Comprehensive Look at jQuery Selectors*. 2014, [Online; navštíveno 30.04.2018].
URL https://link.springer.com/article/10.3758/s13428-014-0471-1

[10] Wang, K.: *MIT License (Expat)*. [Online; navštíveno 30.04.2018].
URL https://tldrlegal.com/license/mit-license

[11] World Wide Web Consortium: *HTML 5.2: 1. Introduction*. [Online; navštíveno 28.04.2018].
URL https://www.w3.org/TR/html5/introduction.html

Příloha A

Obsah paměťového média

- adresář *danceSys* - implementovaný informační systém
- soubor *init-database.sql* - skript pro inicializaci databáze
- soubor *README.md* - instalační pokyny
- adresář *source-latex* - zdrojové soubory technické zprávy pro LaTeX
- soubor *xopich00-BP.pdf* - technická zpráva